Hall Ticket Number:

24CB104/24CS104/24DS104/24IT104

I/IV B.Tech (Regular) DEGREE EXAMINATION

December, 2024 — **Common to CB, CSE, DS & IT Branches**

First Semester — **Introduction to Programming**

Time: Three Hours — **Maximum: 60 Marks**

Answer question 1 compulsorily. **(12X1 = 12 Marks)**

Answer one question from each unit. **(4X12 = 48 Marks)**

			CO	BL	M
1	a)	Define Algorithm and a flow chart.	CO1	L1	1M
	b)	Differentiate compiler and an Interpreter.	CO1	L1	1M
	c)	Write the associativity and priority for the following operators. *, ||, + ,( )	CO1	L1	1M
	d)	List the various format specifiers used in the scanf function.	CO2	L1	1M
	e)	What are the differences between break and continue statements?	CO2	L1	1M
	f)	Write the syntax and working of a do-while loop.	CO2	L1	1M
	g)	Define an Array.	CO3	L1	1M
	h)	What is the scope of a variable?	CO3	L1	1M
	i)	List the different types of storage classes in C language.	CO3	L1	1M
	j)	What is the purpose of main function in a c program.	CO4	L1	1M
	k)	Recall the difference between iteration and recursion.	CO4	L1	1M
	l)	What is a variable length argument list?	CO4	L1	1M
		Unit –I			
2	a)	Summarize the popular features of the C programming language and write an algorithm to find the largest number among three inputs.	CO1	L2	6M
	b)	List and explain the steps involved in the Software life cycle.	CO1	L2	6M
		(OR)			
3	a)	Discuss about the following operators in C language. i) Arithmetic operators ii) Logical operators	CO1	L4	6M
	b)	Create a program to compute the total marks of a student using variables and expressions.	CO1	L2	6M
		Unit –II			
4	a)	Explain all formatted and unformatted input functions with suitable examples.	CO2	L2	6M
	b)	Develop a program to determine whether a given year is a leap year or not.	CO2	L3	6M
		(OR)			
5	a)	Explain various looping statements.	CO2	L2	6M
	b)	Implement a program with a switch statement to display the name of the day based on a numeric input (1-7).	CO2	L3	6M
		Unit –III			
6	a)	Explain the string handling functions that can be used with string.h header file.	CO3	L2	6M
	b)	Construct a program to read values into two matrices and perform sum of the matrices and print the result.	CO3	L3	6M
		(OR)			
7	a)	Explain different types of storage classes used in C with examples.	CO3	L2	8M
	b)	Develop a program using extern to share a variable between two different functions.	CO3	L3	4M
		Unit –IV			
8	a)	Explain the various types of user defined functions with suitable examples.	CO4	L2	8M
	b)	Write a c program to using functions to print Fibonacci series up to a given number.	CO4	L3	4M
		(OR)			
9	a)	Explain the behaviour of a function when parameters are passed by value versus passed by reference.	CO4	L2	6M
	b)	Develop a recursive function to find the factorial of a given number.	CO4	L3	6M

a) Define Algorithm and a flow chart. **1M**

Algorithm: A step-by-step procedure to solve a problem.

Flowchart: A graphical representation of an algorithm using symbols.

b) Differentiate compiler and an Interpreter. **1M**

Compiler: Translates the entire program into machine code at once.

Interpreter: Translates and executes the program line by line.

c) Write the associativity and priority for the following operators. *, ||, + ,( ) **1M**

Operator Associativity and Priority:

- **( ):** Highest Priority, Left to Right Associativity
- ***:** High Priority, Left to Right Associativity
- **+:** Medium Priority, Left to Right Associativity
- **||:** Low Priority, Left to Right Associativity

d) List the various format specifiers used in the scanf function. **1M**

Format Specifiers in scanf():

- %d: Integer
- %f: Float
- %c: Character
- %s: String
- %lf: Double

e) What are the differences between break and continue statements? **1M**

break: Exits the current loop entirely.

continue: Skips the current iteration and proceeds to the next.

f) Write the syntax and working of a do-while loop. **1M**

Syntax:

```
do
{
  // statements
} while (condition);
```

Working: Executes the block of code at least once, then checks the condition. If true, repeats the block.

g) Define an Array. **1M**

Array: A collection of elements of the same data type stored in contiguous memory locations.

h) What is the scope of a variable? **1M**

Scope of a variable: The region of the program where a variable can be accessed and used.

i) List the different types of storage classes in C language. **1M**

Storage Classes in C:

- auto
- register
- static
- extern

j) What is the purpose of main function in a c program. 1M
Main function: The entry point of a C program. Execution begins from here.

k) Recall the difference between iteration and recursion. 1M
Iteration: Repeated execution of a block of code using loops (for, while, do-while).
Recursion: A function calling itself directly or indirectly.

l) What is a variable length argument list? 1M
Variable Length Argument List: A function that can accept a variable number of arguments. (e.g., printf)

UNIT_I

2. a) Summarize the popular features of the C programming language and write an algorithm to find the largest number among three inputs. 6M

Features of C language----3M Algorithm----3M

Popular Features of C Programming Language:

- **Efficiency:** C is known for its speed and efficiency, making it suitable for system programming and performance-critical applications.
- **Portability:** C programs can be easily adapted to run on different operating systems and hardware platforms with minimal changes.
- **Modularity:** C supports modular programming, allowing code to be organized into reusable functions and modules.
- **Rich Standard Library:** C provides a comprehensive set of built-in functions for various tasks, including input/output, string manipulation, and mathematical operations.
- **Low-Level Access:** C offers direct access to memory, enabling efficient memory management and control over hardware resources.
- **Pointer Support:** C's pointer capabilities allow for flexible memory manipulation and data structure implementation.

Algorithm to Find the Largest Number Among Three Inputs:

1. **Start**
2. **Declare three variables:** num1, num2, num3 to store the input numbers.
3. **Read the values of num1, num2, and num3 from the user.**
4. **Initialize a variable largest with the value of num1.**
5. **Compare largest with num2.**
    - If num2 is greater than largest, update largest with the value of num2.
6. **Compare largest with num3.**
    - If num3 is greater than largest, update largest with the value of num3.
7. **Print the value of largest as the largest number.**
8. **End**

b) List and explain the steps involved in the Software life cycle. 6M

Listing the steps ----2M Explanation---4M

The Software Development Life Cycle (SDLC) is a structured framework used to guide the process of creating, testing, and deploying software. It ensures a systematic and organized approach to software development, helping to minimize risks, improve quality, and meet project deadlines.

Here are the key steps involved in the SDLC:

1. **Planning and Requirement Analysis:**
    - **Define project goals and objectives:** Clearly outline the purpose and scope of the software project.
    - **Gather user requirements:** Identify the needs and expectations of the end-users through interviews, surveys, and other methods.
    - **Feasibility study:** Assess the technical, economic, and operational feasibility of the project.
    - **Create a project plan:** Outline the project schedule, budget, resources, and risk management strategies.
2. **Design:**
    - **System design:** Define the overall architecture of the software system, including hardware and software components.
    - **Software design:** Create detailed designs for modules, classes, and user interfaces.
    - **Database design:** Design the database schema to store and manage data efficiently.
3. **Implementation (Coding):**
    - **Write code:** Develop the software using the chosen programming language and tools.
    - **Code reviews:** Conduct peer reviews to identify and fix coding errors and inconsistencies.
    - **Unit testing:** Test individual components of the software to ensure they function correctly.
4. **Testing:**
    - **Integration testing:** Test the interaction between different components of the software.
    - **System testing:** Test the entire software system to ensure it meets the specified requirements.
    - **User acceptance testing (UAT):** Allow end-users to test the software in a real-world environment.
5. **Deployment:**
    - **Release management:** Plan and execute the software release to production environments.
    - **Installation and configuration:** Install the software on target systems and configure it according to user needs.
    - **User training:** Provide training to end-users on how to use the software effectively.
6. **Maintenance:**
    - **Bug fixes:** Address any bugs or defects that are discovered after deployment.
    - **Enhancements:** Add new features and functionalities to the software based on user feedback and changing requirements.
    - **Support:** Provide technical support to end-users to resolve any issues they may encounter.

3. a) Discuss about the following operators in C language. 6M

i) Arithmetic operators -----3M ii) Logical operators------3M

Operators in C Language

i) Arithmetic Operators

Arithmetic operators are used to perform mathematical calculations on numerical values. The common arithmetic operators in C are:

- **Addition (+):** Adds two operands.

  int sum = 5 + 3; // sum will be 8

- **Subtraction (-):** Subtracts the second operand from the first.

  int difference = 10 - 4; // difference will be 6

- **Multiplication (*):** Multiplies two operands.

  int product = 2 * 6; // product will be 12

- **Division (/):** Divides the first operand by the second.

  int quotient = 15 / 3; // quotient will be 5

- **Modulo (%):** Gives the remainder of the division.

  int remainder = 17 % 5; // remainder will be 2

ii) Logical Operators

Logical operators are used to combine conditions or expressions and determine the overall truth value. The common logical operators in C are:

- **Logical AND (&&):** Returns true (1) if both operands are true, otherwise returns false (0).

  ```
  if (age > 18 && age < 65) {
     // code to be executed if age is between 18 and 65
  }
  ```

- **Logical OR (||):** Returns true (1) if at least one of the operands is true, otherwise returns false (0).

  ```
  if (is_member || discount_applied) {
     // code to be executed if either is_member or discount_applied is true
  }
  ```

- **Logical NOT (!):** Reverses the logical state of its operand. If the operand is true, it becomes false, and vice versa.

  ```
  if (!is_logged_in) {
     // code to be executed if is_logged_in is false
  }
  ```

Logical operators are often used in conditional statements (if, else if, else) and loops (while, for) to control the flow of execution based on multiple conditions.

b) Create a program to compute the total marks of a student using variables and

expressions. **6M**

```
#include <stdio.h>

int main() {

   int subject1, subject2, subject3;

   int total_marks;

   // Get marks for each subject from the user

   printf("Enter marks for subject 1: ");

   scanf("%d", &subject1);

   printf("Enter marks for subject 2: ");

   scanf("%d", &subject2);

   printf("Enter marks for subject 3: ");

   scanf("%d", &subject3);

   // Calculate total marks

   total_marks = subject1 + subject2 + subject3;

   // Print the total marks

   printf("Total marks: %d\n", total_marks);

   return 0;

}
```

UNIT_II

4. a) Explain all formatted and unformatted input functions with suitable examples. 6M

Formatted----3M Unformatted input functions---3M

Formatted Input Functions

- **scanf():** This is the most common formatted input function. It reads data from the standard input (usually the keyboard) and stores it in variables according to specified format specifiers.
  Example:

```
#include <stdio.h>
int main() {
   int age;
   float weight;
   char name[50];
```

```
    printf("Enter your age: ");
    scanf("%d", &age);

    printf("Enter your weight: ");
    scanf("%f", &weight);

    printf("Enter your name: ");
    scanf("%s", name);

    printf("Age: %d, Weight: %.2f, Name: %s\n", age, weight, name);
    return 0;
}
```

  - %d: Format specifier for reading an integer.
  - %f: Format specifier for reading a floating-point number.
  - %s: Format specifier for reading a string.
  - &: Address-of operator, used to pass the address of the variable to scanf().
- **fscanf():** Similar to scanf(), but reads data from a specified file stream instead of standard input.

  Example:

```
#include <stdio.h>

int main() {
    FILE *fp;
    int age;

    fp = fopen("data.txt", "r");
    if (fp == NULL) {
        printf("Error opening file.\n");
        return 1;
    }

    fscanf(fp, "%d", &age);
    printf("Age from file: %d\n", age);

    fclose(fp);
    return 0;
}
```

Unformatted Input Functions

- **getchar():** Reads a single character from the standard input.
  Example:

```
#include <stdio.h>

int main() {
    char ch;

    printf("Enter a character: ");
```

```
    ch = getchar();

    printf("You entered: %c\n", ch);
    return 0;
}
```

- **gets():** Reads a line of text from the standard input, including spaces. **Note:** gets() is considered unsafe because it doesn't check for buffer overflows. It's generally recommended to use fgets() instead.

  Example:

```
#include <stdio.h>

int main() {
    char str[100];

    printf("Enter a line of text: ");
    gets(str);

    printf("You entered: %s\n", str);
    return 0;
}
```

- **fgets():** Reads a line of text from a specified stream, including spaces, and stores it in a character array. It's safer than gets() because you can specify the maximum number of characters to read, preventing buffer overflows.

  Example:

```
#include <stdio.h>

int main() {
    char str[100];

    printf("Enter a line of text: ");
    fgets(str, 100, stdin);

    printf("You entered: %s\n", str);
    return 0;
}
```

b) Develop a program to determine whether a given year is a leap year or not. 6M

```
#include <stdio.h>
int main() {
    int year;
    printf("Enter a year: ");
    scanf("%d", &year);

    if ((year % 4 == 0 && year % 100 != 0) || (year % 400 == 0)) {
        printf("%d is a leap year.\n", year);
```

```
    } else {
        printf("%d is not a leap year.\n", year);
    }
    return 0;
}
```

5. a) Explain various looping statements. **6M**

Each loop carries ---2M

Looping Statements in C

Looping statements in C allow you to repeatedly execute a block of code as long as a certain condition is met. This significantly reduces code redundancy and makes your programs more efficient. C provides three primary looping constructs:

1. for Loop

- **Syntax:**

```
for (initialization; condition; increment/decrement) {
   // Code to be executed repeatedly
}
```

- **Example:**

```
for (int i = 0; i < 10; i++) {
   printf("%d ", i); // Prints numbers from 0 to 9
}
```

2. while Loop

- **Syntax:**

```
while (condition) {
   // Code to be executed repeatedly
}
```

- **Explanation:**
  - The while loop first checks the condition. If it's true, the loop body is executed. After executing the loop body, the condition is checked again. This process continues as long as the condition remains true.
- **Example:**

```
int i = 0;
while (i < 5) {
   printf("%d ", i); // Prints numbers from 0 to 4
   i++;
}
```

3. do-while Loop

- **Syntax:**

```
do {
   // Code to be executed repeatedly
} while (condition);
```

- **Example:**

```
int i = 0;
do {
   printf("%d ", i); // Prints numbers from 0 to 4
   i++;
} while (i < 5);
```

b) Implement a program with a switch statement to display the name of the day based on a numeric input (1-7). 6M

```
#include <stdio.h>
int main() {
   int day;
   printf("Enter a number (1-7): ");
   scanf("%d", &day);

   switch (day) {
      case 1:
         printf("Sunday\n");
         break;
      case 2:
         printf("Monday\n");
         break;
      case 3:
         printf("Tuesday\n");
         break;
      case 4:
         printf("Wednesday\n");
         break;
      case 5:
         printf("Thursday\n");
         break;
      case 6:
         printf("Friday\n");
         break;
      case 7:
         printf("Saturday\n");
         break;
      default:
         printf("Invalid input!\n");
   }
   return 0;
}
```

UNIT_III

6.a) Explain the string handling functions that can be used with string.h header file. 6M

String handling functions any six can be considered------6M

The string.h header file in C provides a rich set of functions for manipulating strings. Here are some of the most common ones:

1. String Length:

- **strlen(str):**
  - Calculates and returns the length of the string str.
  - The length is determined by counting the number of characters in the string, excluding the null character (\0).

2. String Copying:

- **strcpy(dest, src):**
  - Copies the entire string src to the destination string dest.
  - **Important:** Ensure dest has enough space to accommodate the entire src string.
- **strncpy(dest, src, n):**
  - Copies at most n characters from src to dest.
  - If src has fewer than n characters, only those characters are copied.
  - If src has more than n characters, only the first n characters are copied.

3. String Concatenation:

- **strcat(dest, src):**
  - Appends the string src to the end of the string dest.
  - **Important:** dest must have enough space to hold both strings.
- **strncat(dest, src, n):**
  - Appends at most n characters from src to the end of dest.

4. String Comparison:

- **strcmp(str1, str2):**
  - Compares two strings lexicographically.
  - Returns:
    - 0 if str1 and str2 are equal.
    - A negative value if str1 is less than str2.
    - A positive value if str1 is greater than str2.
- **strncmp(str1, str2, n):**
  - Compares at most n characters of str1 and str2.

5. String Searching:

- **strchr(str, ch):**
  - Finds the first occurrence of the character ch within the string str.
  - Returns a pointer to the first occurrence of ch, or NULL if ch is not found.
- **strrchr(str, ch):**
  - Finds the last occurrence of the character ch within the string str.
- **strstr(str1, str2):**
  - Finds the first occurrence of the substring str2 within the string str1.

- Returns a pointer to the beginning of the first occurrence of str2 within str1, or NULL if str2 is not found.

6. String Conversion:

- **strupr(str):**
  - Converts all lowercase letters in str to uppercase.
- **strlwr(str):**
  - Converts all uppercase letters in str to lowercase.

Example:

```
#include <stdio.h>
#include <string.h>

int main() {
   char str1[100] = "Hello";
   char str2[] = " World!";

   // Concatenate strings
   strcat(str1, str2);
   printf("Concatenated string: %s\n", str1);

   // Compare strings
   if (strcmp(str1, "Hello World!") == 0) {
      printf("Strings are equal.\n");
   }

   return 0;
}
```

b) Construct a program to read values into two matrices and perform sum of the matrices and print the result. 6M

```
#include <stdio.h>

int main() {
   int rows, cols;

   // Get dimensions of the matrices from the user
   printf("Enter the number of rows and columns: ");
   scanf("%d %d", &rows, &cols);

   // Declare matrices
   int matrix1[rows][cols], matrix2[rows][cols], sum[rows][cols];

   // Read values for matrix1
   printf("\nEnter elements of matrix 1:\n");
   for (int i = 0; i < rows; i++) {
      for (int j = 0; j < cols; j++) {
         scanf("%d", &matrix1[i][j]);
      }
```

```
    }

    // Read values for matrix2
    printf("\nEnter elements of matrix 2:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            scanf("%d", &matrix2[i][j]);
        }
    }

    // Calculate sum of matrices
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            sum[i][j] = matrix1[i][j] + matrix2[i][j];
        }
    }

    // Print the sum matrix
    printf("\nSum of matrices:\n");
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            printf("%d ", sum[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

7.a) Explain different types of storage classes used in C with examples. 8M

Four Storge Classes each ----2M

1. Auto

- **Default storage class:** If no storage class is specified, it's automatically auto.
- **Scope:** Local to the block where it's declared.
- **Lifetime:** Exists only within the block where it's declared. Once the block ends, the variable is destroyed and its value is lost.
- **Example:**

```
void myFunction() {
    int x = 10; // 'x' is an auto variable
    // ...
}
```

2. Register

- **Suggests to the compiler:** Store the variable in a CPU register for faster access.
- **Scope and Lifetime:** Same as auto.
- **Note:** The compiler may or may not honor this request.
- **Example:**

```
void myFunction() {
   register int count = 0;
   // ...
}
```

3. Static

- **Scope:**
  - If declared within a function: Local to the function, but retains its value between function calls.
  - If declared outside a function: Global scope (visible to all functions in the file).
- **Lifetime:**
  - For local static variables: Exists throughout the program's execution.
  - For global static variables: Exists throughout the program's execution.
- **Example:**

```
int myFunction() {
   static int count = 0; // 'count' retains its value between calls
   count++;
   return count;
}
```

4. Extern

- **Declares a variable that is defined elsewhere:** Used to access a global variable declared in another file.
- **Scope:** Global (if declared outside a function) or local to the block (if declared within a function).
- **Lifetime:** Exists throughout the program's execution.
- **Example:**

In file1.c:

```
int global_var = 10;
```

In file2.c:

```
#include <stdio.h>
extern int global_var; // Declares that global_var is defined elsewhere

int main() {
   printf("%d\n", global_var);
   return 0;
}
```

b) Develop a program using extern to share a variable between two different functions. 4M

```
// file1.c

#include <stdio.h>

int global_count; // Declare global_count

void increment() {
   global_count++;
}

// file2.c

#include <stdio.h>

extern int global_count; // Declare global_count as external

int main() {
   increment();
   printf("Global count: %d\n", global_count);
   return 0;
}
```

8.a) Explain the various types of user defined functions with suitable examples. 8M

Four different types each ----2M

1. Function with No Arguments and No Return Value

- **Purpose:** Performs a specific task without needing input or producing an output.
- **Syntax:**

```
void function_name() {
   // Code to be executed
}
```

- **Example:**

```
void greet() {
 printf("Hello, world!\n");
}
```

2. Function with Arguments but No Return Value

- **Purpose:** Performs a task based on input values but doesn't produce a direct result.
- **Syntax:**

```
void function_name(data_type arg1, data_type arg2, ...) {
  // Code to be executed
}
```

Example:

```
void display(int num) {
  printf("The number is: %d\n", num);
```

```
}
```

3. Function with Arguments and Return Value

- **Purpose:** Performs a task, uses input values, and produces a result.
- **Syntax:**

```
return_type function_name(data_type arg1, data_type arg2, ...) {
   // Code to be executed
   return value;
}
```

- **Example:**

```
int add(int a, int b) {
   int sum = a + b;
   return sum;
}
```

4. Function with No Arguments but Return Value

- **Purpose:** Performs a task without needing input but produces a result.
- **Syntax:**

```
return_type function_name() {
   // Code to be executed
  return value;
}
```

- **Example:**

```
int generateRandomNumber() {
// Generate a random number using a suitable function (e.g., rand())
return random_number;
}
```

b) Write a c program to using functions to print Fibonacci series up to a given number. 4M

```
#include <stdio.h>

// Function to print Fibonacci series up to a given number
void printFibonacci(int n) {
   int first = 0, second = 1, next, i;

   printf("Fibonacci Series:\n");

   for (i = 0; i <= n; ++i) {
      printf("%d ", first);
      next = first + second;
      first = second;
      second = next;
   }
```

```
}

int main() {
    int limit;

    printf("Enter the limit: ");
    scanf("%d", &limit);

    printFibonacci(limit);

    return 0;
}
```

9.a) Explain the behaviour of a function when parameters are passed by value versus passed by reference. 6M

Pass by value ----3M Pass by Reference----3M

1. Pass by Value

- **Mechanism:**
  - When a parameter is passed by value, a copy of the actual argument is made and passed to the function.
  - The function operates on this copy, leaving the original argument in the calling function unchanged.
- **Example:**

```
void square(int num) { // num receives a copy of the argument
  num = num * num;
  printf("Square of num inside function: %d\n", num);
}

int main()
 {
    int x = 5;
    square(x);
    printf("Square of x in main: %d\n", x); // x remains unchanged
   return 0;
}
```

2. Pass by Reference

- **Mechanism:**
  - When a parameter is passed by reference, the memory address of the argument is passed to the function.
  - The function directly operates on the original argument in the calling function.
- **Example:**

```
void square(int *num) { // num receives the address of the argument
   *num = *num * *num;
 printf("Square of num inside function: %d\n", *num);
}
```

```
int main() {
  int x = 5;
  square(&x); // Pass the address of x
 printf("Square of x in main: %d\n", x); // x is now modified
  return 0;
}
```

b) Develop a recursive function to find the factorial of a given number. **6M**

```
#include <stdio.h>

int factorial (int n) {
   if (n == 0) {
      return 1; // Base case: factorial of 0 is 1
   } else {
      return n * factorial(n - 1); // Recursive step
   }
}

int main() {
   int num;

   printf("Enter a number: ");
   scanf("%d", &num);

   if (num < 0) {
      printf("Factorial is not defined for negative numbers.\n");
   } else {
      int result = factorial(num);
      printf("Factorial of %d is %d\n", num, result);
   }

   return 0;
}
```

Scheme prepared by Signature of the HOD, IT Dept.

Paper Evaluators:

S.No	Name Of the College	Name of the Faculty	Signature